Relatório: Diagnóstico de circuitos lógicos digitais por tabela verdade

Rodrigo Teixeira de Melo
Mestrado Acadêmico em Ciência da Computação
Universidade Estadual do Ceará

***Resumo*—Este trabalho divide-se em três etapas, correspondentes a cada questão do trabalho de programação 01 do curso de Inteligência Artificial do Mestrado Acadêmico de Ciência da Computação. O trabalho explica como, e até onde foi resolvido o problema proposto na disciplina, a estrutura do documento é dividida em cinco itens.**

I. Definição do problema

Os circuitos digitais podem falhar. Um circuito digital combinatório pode ser especificado de duas formas: pela sua função lógica ou pela sua tabela verdade. Dizemos que um circuito lógico contém uma ou mais falhas quando a função lógica que ele executa de fato não corresponde aquela da especificação. A forma de observar que um circuito lógico combinatório está com falhas é verificando que a tabela verdade do seu funcionamento real não corresponde a tabela verdade da função lógica especificada. O trabalho proposto sugere que uma porta lógica só possui quatro tipos de falha: a saída ou uma entrada (ou mais de uma entrada) pode estar "presaEm0" ou "presaEm1", entretanto, por limitações no momento da implementação, este trabalho considera que apenas as *entradas* das portas lógicas podem estar "presaEm0" ou "presaEm1".

Seguindo as etapas do Capítulo 03 do livro, é definido os conceitos de estado, estado inicial, função sucessor, teste de objetivo e custo de caminho, para as questões A e B. A questão "A" é incluida no problema, visto que gerar os bits para calcular a tabela verdade, de forma não repetida, também pode ser visualizada como problema combinatório envolvendo busca.

- **Estado:** para a questão A, onde o objetivo é a geração das tabelas verdade com e sem falhas, um estado pode ser definido como uma combinação de valores para cada linha do circuito, que liga cada entrada das portas. Sendo assim uma tabela verdade é gerada a partir de 2^n combinações de estados, onde n é o número de entradas do circuito (não confundir com entrada de portas lógicas). Já para o problema do diagnóstico, questão B, um estado pode ser definido como uma atribuição de n falhas para n entradas de portas lógicas - visto que neste trabalho não foi implementada rotina para teste de falhas em saídas -, entretanto, só foi criada a rotina de diagnóstico para uma falha.
- **Estado inicial:** Para questão A, qualquer combinação de bits 0 ou 1, que representam as entradas do circuito. Questão B, o estado inicial é uma projeção do circuito ideal, ou seja, sem falhas.
- **Função sucessor:** Questão A, gera uma nova combinação válida de bits para teste do circuito. Questão B, gera uma combinação de falhas nas entradas das portas lógicas.
- **Teste de objetivo:** Questão A, verificar se foram gerados 2^n combinações de bits, com as devidas operações calculadas. Questão B, Verificar se a tabela verdade, calculada a partir do circuito com a falha induzida é equivalência lógica do circuito com falhas, fornecido pelo usuári0
- **Custo de Caminho:** Cada passo custa 1, então o custo do caminho é o número de passos.

Dada a breve introdução acerca do domínio do problema, temos as tarefas a serem resolvidas:

A. *Implementação A*

Dado um arquivo de texto com a descrição de um circuito lógico combinatório, escrever em um arquivo de saída:

- A função lógica delimitada por parênteses;
- A tabela verdade do circuito sem falhas;
- A tabela verdade do circuito defeituoso.

No qual os itens acima podem ser implementados separadamente ou de forma integrada.

B. *Implementação B*

Escrever um procedimento de busca exaustiva, que lê como entrada as tabelas verdade do circuito sem falha e do circuito com falhas, gerados pelo item I-A, e escreve em um arquivo de saída todas as combinações de n falhas que correspondem ao comportamento defeituoso do circuito.

C. *Implementação C*

1) : Dado um circuito com duas saídas, averiguar o conjunto combinações de falhas para cada saída conforme item I-B, é lógico pensar que se uma falha está presente no circuito, ela será detectada em cada uma das saídas. Com isso em mente, escrever uma rotina para averiguar as falhas em cada uma das saídas, e apontar as falhas que são interseção nas saídas do circuito, conforme a Equação 1:

$$F = S_{f1}^{(cj)} \bigcap S_{f2}^{(cj)} \bigcap ... \bigcap S_{fn}^{(cj)} \tag{1}$$

Onde $S_{fn}^{(cj)}$ é a saída S, com uma falha fn para um determinado circuito cj, e F é a "melhor" falha candidata, visto que esta encontra-se presente em todos os conjuntos de falhas de saídas isoladas.

2) : Implementar busca em grafos em um circuito maior, composto dos quatro circuitos especificados para o problema.

II. SOLUÇÕES PROPOSTAS

Nesta seção são abordadas as técnicas de solução para o problema, que podem, em alguns casos, diferir ligeiramente do que foi abordado em sala de aula, em virtude de improvisos, limitações de linguagem de programação e de tempo, etc.

A. *Solução para a questão A*

A questão A consiste - em resumo - em calcular a tabela verdade do circuito sem falhas, e com falhas, sendo o circuito descrito em um arquivo de texto, bem como as falhas e apresentar sua descrição formal delimitada por parenteses.

A estratégia adotada para tal, foi o uso de um *parser*[1] de texto, específico para lógica booleana, o *parser* analisa sintáticamente o circuito descrito no texto, que deve obedecer uma sequência rigorosa explicada a seguir:

```
<saída>=(entrada <operação> entrada)
```

O circuito descrito no texto deve conter uma saída, imediatamente seguida por um sinal de igualdade (sem espaços em branco), que por sua vez é imediatamente seguido pela descrição das portas, em caso de uso de parênteses, é obrigatório que um parêntese, quando aberto, seja fechado, caso contrário é acusado erro de desbalanceamento da expressão.

É perfeitamente possível que uma porta possua mais de duas entradas, e várias operações delimitadas por parênteses, como no exemplo:

```
S=(A and B and C and D) or (C xor E)
```

É possível também aninhar parênteses na descrição do circuito, no exemplo a seguir, é demonstrado o uso de parênteses aninhados, onde cada operação dentro de um cunjunto de parênteses é uma porta do circuito:

```
S=(((A xor B) and C) or not(D))
```

A implementação realizada para o problema I-A, também permite que diversas saídas para o mesmo circuito sejam descritas em um mesmo arquivo de texto, segue o exemplo do circuito 01 (no PDF enviado por e-mail) proposto para este trabalho de implementação:

```
S=((A xor B) xor C)
Cout=((A xor B) and C) or (A and B))
```

As saídas devem ser descritas uma abaixo da outra, utilizando apenas quebra de linha, cada linha deve obedecer os critérios de construção já explicados e por razões de simplificação de implementação, todas as entradas devem possuir apenas uma letra, assim, Cin no circuito vira apenas C.

A tabela verdade é calculada usando as estruturas fornecidas pelo *parser*, com as funções nativas da linguagem Python, ou seja, o *parser* converte expressões como:

```
(A xor B)
```

em

```
A ^ B
```

Onde "^" é o operador XOR na linguagem Python, permitindo assim que as entradas sejam calculadas como funções matemáticas.

Após isso, é gerado o conjunto de combinações de entradas possíveis para cada entrada de uma porta. Para a porta definida como "(A xor B)" temos as entradas A e B, no qual cada uma pode assumir o valor 0 ou 1, sendo assim temos 2^n combinações possíveis de entrada, onde n é a quantidade de variáveis de entrada. Neste exemplo temos $2^2 = 4$ combinações possíveis, se a porta fosse definida por "(A and B and C)" teríamos $2^3 = 8$ combinações possíveis de entradas.

É possível então observar que gerar tabela verdade para um circuito é uma tarefa de ordem exponencial em espaço de armazenamento, e é possível que esta também seja exponencial em tempo.

Para contornar a natureza que este problema tem de ser exponencial, é adotada uma técnica *linear* em tempo de execução, para geração de todas as combinações possíveis de entrada, exemplificada pela Equação 2:

$$X = \sum_{i=2^n}^{1} bin(2^n - i) \tag{2}$$

Onde X é o conjunto de todas as combinações possíveis para n entradas, e a função bin, converte um número decimal para binário, é importante ressaltar que são considerados apenas os n valores finais obtidos da equação em questão.

Os resultados da Equação 2 para $n = 3$ são demonstrados na Tabela I.

$i =$	Resultado
8	000
7	001
6	010
5	011
4	100
3	101
2	110
1	111

Tabela I. RESULTADOS DA EQUAÇÃO 2 PARA $n = 3$.

Os valores gerados para cada iteração i são então divididos e atribuidos a cada entrada do circuito, ou seja, o valor 010 significa que $A = 0$, $B = 1$, $C = 0$, gerando então uma matriz com todas as combinações de entradas possíveis para qualquer circuito de n entradas.

As operações de cada porta (and, nand, or, nor, xor, nxor, not) são calculadas ao buscar o valor de combinações de entradas que correspondem a operação, isto é, na operação "(A and B)", para cada linha da Tabela I, busca-se o valor para A e para B, e efetua-se a operação *and* entre estes valores, o resultado é guardado e utilizado nas operações seguintes, se necessário.

[1] *Parsing*, ou **análise sintática**, é o processo de analisar uma sequência de entrada, para determinar sua estrutura gramatical segundo uma determinada gramática formal. Fonte: Wikipedia.

Desta forma, é possível observar que gerar todas as combinações possíveis para um número n de entradas é de complexidade linear, enquanto que calcular o circuito baseado nestas combinações, é (simplificadamente) de ordem quadrática, visto que é necessário apenas realizar todas as operações de portas ao percorrer as colunas, utilizando todas as combinações possíveis de entradas, presentes nas linhas.

A descrição das falhas no circuito são feitas trocando uma entrada de uma porta pela expressão da falha correspondente. As expressões que indicam falhas nas entradas são:

- p0 $\rightarrow$ Preso em zero
- p1 $\rightarrow$ Preso em um

Assim, considerando a saída S do primeiro circuito proposto:

```
S=((A xor B) xor C)
```

Para introduzir uma falha "PresoEm0" na *segunda entrada* da *primeira porta xor*, o circuito ficaria descrito da seguinte forma:

```
S=((A xor p0) xor C)
```

E para introduzir uma falha "PresoEm1" na *primeira entrada* da *primeira porta xor* o circuito seria descrito como:

```
S=((p1 xor B) xor C)
```

No algoritmo, os valores p0 e p1 são trocados por 0 e 1, respectivamente, para que seu valor verdade fique sempre atribuído desta forma quando for realizada as operações de resolução da tabela verdade.

Obs: Para simplificação da implementação, e por limite de tempo, não foram escritas rotinas para introduzir falhas nas saídas de portas, e nem para descrever os circuitos de forma "recorrente".

As tabelas geradas como resolução da implementação do item A são salvas em um arquivo "saída.txt", e serão apresentados em uma seção apropriada.

B. *Solução para a questão B*

Para a realização do diagnóstico do circuito, a partir da tabela verdade do circuito perfeito e do circuito com defeito, foi criada uma rotina que introduz em cada entrada do circuito, uma falha *PresaEm0* ou *PresaEm1* por vez.

Sendo assim, um circuito com 6 entradas, possui $6x2 = 12$ combinações de uma falha possíveis. Foi iniciada a implementação do algoritmo para introduzir combinações de n falhas, entretanto o mesmo não ficou pronto a tempo da entrega do trabalho.

Ao fim da geração da falha em uma iteração, é calculada a tabela verdade para o circuito com esta nova falha, e então a tabela verdade gerada nesta iteração, é comparada a tabela com falha gerada pelo exercício da questão A, se as duas tabelas forem equivalências lógicas, então o circuito com falha que gera esta tabela é guardado no arquivo de texto, como uma das causas para o problema do circuito, o pseudocódigo a seguir exemplifica o procedimento.

```
circComFalha = tabelaVerdade(exprComFalha)
for entrada in exprComFalha do
    novaExpr = gerarFalhas(entrada,exprComFalha)
    novoCircComFalha = tabelaVerdade(novaExpr)
    if cirComFalha is equivLogica(novoCircComFalha) then
        salvarNoArquivo(novaExpr)
    end if
end for
```

Desta forma, são geradas todas as combinações de uma falha, e são apontadas todas as possíveis causas para o circuito com comportamento defeituoso.

C. *Solução para a questão C*

1) : A implementação para a solução da questão C-1 foi iniciada, mas não ficou pronta a tempo da data da entrega, então não será especificada.

2) : A junção dos 04 circuitos foi realizada conforme descrita para a tarefa, as entradas G5, G2, G4, G3 e G1 do circuito C2 recebem, respectivamente, as saídas O1, O2, 1, 0 e S, dos circuitos C3, C4 e C1.

O método para combinações de falhas adotado consiste em percorrer cada entrada de cada porta, atribuindo as falhas uma por vez. A complexidade de espaço é exponencial, visto que não há como tratar isso, pois o trabalho envolve tabelas verdade. Para exemplificar, uma das três saídas do circuito 5, fusão dos demais circuitos, possui em sua tabela verdade sem falhas, mais de 12 mil linhas! Entretanto, toda diagnóstico leva pouco mais de 20 segundos.

Não foi possível gerar combinações de n falhas para o circuito em questão. Porém a implementação deste trabalho suporta calcular qualquer circuito com um número variável de entradas e saídas, e ainda suporta fazer o diagnóstico para todas as combinações de uma falha em todas as entradas, deste modo, foi possível realizar o diagnóstico preciso do circuito, caso este contenha apenas uma falha em qualquer que seja a entrada.

III. Abordagem

Explicado a forma como cada item do trabalho foi solucionado (os que puderam ser solucionados), a obtenção dos resultados se deu da seguinte forma:

- O programa inteiro consiste de apenas um script "main.py", escrito na linguagem Python;
- Neste script são utilizadas algumas bibliotecas, para funcionalidades cuja sua implementação tornaria o trabalho impossível de entregar a tempo. As bibliotecas são:
    - Numpy, para operações eficientes com matrizes e cálculos matemáticos;
    - Re (Regular Expression ou Regex), para lidar com expressões regulares;
    - Time, para cronometrar a execução do algoritmo;
    - TT, para trabalhar com expressões booleanas;
    - PrettyTable, para gerar tabelas elegantes e mais legíveis para os resultados.
- O algoritmo implementa 04 funções:
    - Uma para gerar todas as combinações de teste, conforme Equação 2;

  - Uma para calcular a tabela verdade do circuito ideal (sem falha) e do circuito defeituoso, a partir da definição de um circuito com falha, salvando o resultado com as duas matrizes e a função delimitada por parênteses no arquivo "saída.txt" (item A do trabalho);
  - Uma função para encontrar ocorrências em *strings*, retornando os índices dessas ocorrências, esta função serve para buscar todo o "alfabeto" de entradas das portas lógicas, para que estas tenham seu valor verdade alterado no momento do cálculo da tabela verdade, esta função é essencial para que seja possível gerar todas as combinações de falhas, e consequentemente, todas as tabelas verdades possíveis, para que o diagnóstico seja realizado;
  - Por último a função de diagnóstico, que utiliza o a função do item acima, compara todas as tabelas verdades de todo o espaço de estados (combinações de uma falha), para um dado circuito, e escreve os erros candidatos em um arquivo "diagnóstico.txt"
- O circuito a ser avaliado é descrito no arquivo "circuito.txt" e este pode conter qualquer quantidade de entradas, de portas, e qualquer quantidade de saídas (separadas por quebra de linha), limitado apenas pela memória e processador do computador. Entretanto, para efeito de legibilidade, é passada apenas uma saída em cada execução por vez. Desta forma, se um circuito tem 2 saídas, para cada saída são gerados 02 arquivos de texto, "saída.txt" e "diagnóstico.txt", de forma a melhorar a legibilidade dos resultados.
- O resultado descrito no arquivo "saída.txt" para apenas uma saída de circuito **com falha**, consiste da função lógica do circuito, onde a variável que indica a falha "p0" ou "p1" é tratada como sendo de fato uma entrada, ou seja, são geradas combinações de 0 e 1 para esta variável, isto permite gerar a tabela verdade para o circuito ideal (sem falhas), e após isso os valores "p0" ou "p1" são de fato presos em 0 ou em 1, conforme sua função, e uma nova tabela verdade é calculada, esta por sua vez representando de fato o circuito com falha.
- Se o circuito for passado ao programa sem a descrição de falhas, ou seja, um circuito normal, o programa gera apenas a tabela verdade para o circuito normal com sua função lógica, e o arquivo de diagnóstico conterá a frase *"Não foram encontradas falhas para o circuito"*
- Cada circuito foi testado com apenas uma execução, todos com 01 falha aleatória descritos manualmente, visto que a função de randomizar 10 falhas não ficou pronta a tempo da entrega do trabalho.

IV. Resultados

Os resultados de cada análise, conforme explicado na seção III, são salvos em 02 arquivos de texto, foram realizados 01 teste, com 01 falha aleatória "p0" ou "p1" em uma entrada aleatória de todos os circuitos testados, inclusive do circuito referente ao item C-II

O arquivo de resultado para o item A ("saída.txt") consiste da seguinte estrutura:

Circuito	Falhas apontadas (uma falha)
1 - Saida S	3
1 - Saída Cout	1
2 - Saída G17	2
2 - Saída G16	2
3 - Saída O1	6
3 - Saída O2	1
4 - Saída 1	1
4 - Saída 2	1
5 - Saída 1	1
5 - Saída 2	1
5 - Saída 3	1

Tabela II. Quantidade de falhas apontadas pelo programa em cada circuito

```
Análise do circuito
Função lógica: <Função do circuito>
Tabela verdade:
+---+---+---+
| A | B | S |
+---+---+---+
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
+---+---+---+
Tempo de execução: <tempo> segundos
```

Sendo que se o circuito possuir falhas, são duas estruturas semelhantes, a primeira para o circuito sem falhas, e a segunda para o circuito com falhas.

O arquivo de diagnóstico consiste da seguinte estrutura, considerando a saída S do circuito 01 com falha "p0" na entrada 2 da segunda porta xor:

```
Circuito: S=((A xor B) xor p0)

-> Falha: ((PortaPresaEm0 xor B) xor XX)
-> Falha: ((A xor PortaPresaEm0) xor XX)
-> Falha: ((A xor B) xor PortaPresaEm0)

Tempo de execução: 0.009 segundos.
```

Vale ressaltar que os itens apontados como *Falha*, são possíveis falhas, cuja a tabela verdade é equivalência lógica do circuito com a falha verdadeira e o valor *XX* é um variável qualquer colocada para substuir os termos que designam uma falha, para que o algoritmo não se confunda durante o processamento.

A Tabela II demonstra a quantidade de falhas encontradas para uma execução do algoritmo em cada circuito, com uma falha.

Os demais resultados para todos os circuitos, não serão apresentados neste relatório, em virtude das tabelas serem grandes demais até para serem apresentadas como apêndices, porém, os resultados estão prontos e vão anexos ao *email*, e se solicitado, os mesmos podem ser

apresentados com maior clareza em sala de aula.
Para todos os circuitos testados, com todas as combinações de uma falha, o maior tempo de processamento é o do diagnóstico da saída 2 do circuito (junção dos demais circuitos), e este tempo é de 26 segundos.
Para a resolução do item A, onde apenas é calculado a tabela verdade (com e sem falha), com exceção da saída 02 do circuito 5, que possui mais de 12 mil linhas, todos os demais circuitos tiveram suas tabelas verdade com e sem falha calculadas em menos de 1 segundo. Alguns circuitos menores, levam milésimos de segundo para processar.

V. Conclusão

Conclui-se que a abordagem adotada, é eficiente em calcular as combinações de uma falha em um determinado circuito, e apontar em muito dos casos, **com exatidão**, qual a porta defeituosa do circuito, apenas analisando suas tabelas verdade.

O algoritmo consegue percorrer exaustivamente todo o **espaço de estados** de combinações de entradas em tempo polinomial, percorrendo uma matriz apenas uma vez, e consegue diagnosticar um problema para uma falha também em tempo polinomial, percorrendo x matrizes apenas uma vez, onde x é o número de combinações de falhas possíveis para as entradas.

Em virtude da complexidade do trabalho, o tempo de implementação não foi suficiente para implementar as rotinas de geração de falhas aleatórias para n falhas de forma automática, entretando a análise para n falhas é possível se o usuário inserir as falhas manualmente no circuito.

Observa-se que quanto maior o número de falhas nas entradas do circuito, menor o tempo de processamento do *algoritmo para gerar as tabelas verdade*. o que mostra que conforme o número de falhas cresce o tamanho da matriz final diminui, por ter menos estados a se percorrer, e assim a complexidade diminui.